

## Lecture 8a:

# Input 2: Declarative input models; “Interactor” (Behavior) Objects in Garnet and Amulet



05-431/631 Software Structures for User Interfaces (SSUI)  
Fall, 2022



# Overview of the “Interactor” Input Model

- Try to provide more support so input handling isn't so difficult
- Make easy things simple and complex things possible
- Based on the "Model-View-Controller" architecture from Smalltalk
  - (Lecture 9)
- True separation of graphics (view) and input handling (controller)
- Also uses idea from [Foley&Wallace 1974] of identifying types of input handlers:
  - move
  - grow
  - rotate
  - text edit
  - gesture
  - select (pick)

[James D. Foley and Victor L. Wallace. “The Art of Natural Graphic Man-Machine Conversation,” *Proceedings of the IEEE*. Apr, 1974. vol. 62, no. 4. pp. 462-471.]



# Innovations

- Identifying primitive "Interactor" objects and correct parameterizations so most direct manipulation UIs can be constructed by re-using built-in objects.
  - Better name might be "Behavior" objects
- Only a few kinds of behaviors, and standard parameters
- Real separation between input and output handling
- Handles all input
  - insides of widgets
  - and for application programs
- + First successful separation of View from Controller in Smalltalk MVC
- + Integration of gestures with conventional interaction.
- + Easier to code because substantial re-use
- + Built-in support for multi-window dragging



# General idea

- Attach **Interactor objects** to a set of graphical objects to handle their input.
- Graphical objects don't handle input
  - No "event methods" in objects
- Instead, define invisible "Interactor" objects and attach them to graphics
- Interactors can operate on multiple objects
- Strategy: pick the right type of Interactor, attach to the objects to be affected, fill in necessary slots of interactor
- Widgets use interactors internally
- Can have multiple interactors on an object (e.g., different mouse buttons)
- Interactors directly set slots of objects using a standard protocol
  - constraints can be used to map those slots into behaviors:
- Details of input events and event processing is hidden
- Used first in Garnet, refined in Amulet.

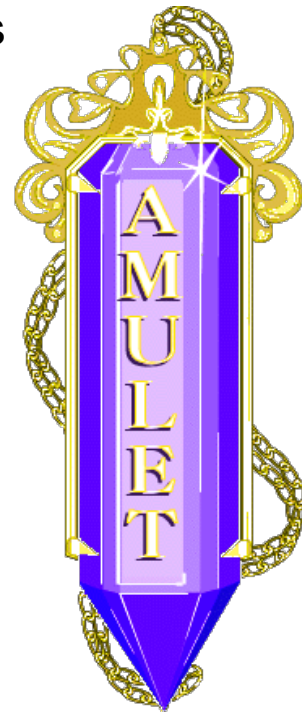
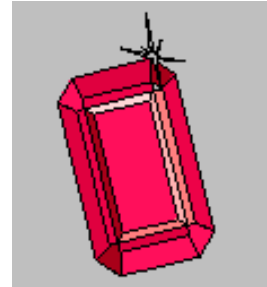
# Flash Catalyst

- Previous product from Adobe
  - Only in CS 5.5
- Also had behaviors that can be attached to graphics and parameterized



## Aside: Garnet and Amulet

- **Garnet:** ([link](#))
  - 1987 to 1994
  - Common Lisp and X11 or Macintosh
  - **G**enerating an **A**malgam of **R**real-time, **N**ovel **E**editors and **T**oolkits
- **Amulet:**
  - 1994 to 1997
  - C++ and X11, Windows or Macintosh
  - **A**utomatic **M**anufacture of **U**sable and **L**earnable **E**editors and **T**oolkits
  - Full Amulet Manual:
    - [http://www.cs.cmu.edu/afs/cs/project/amulet/amulet3/manual/Amulet\\_ManualTOC.doc.html](http://www.cs.cmu.edu/afs/cs/project/amulet/amulet3/manual/Amulet_ManualTOC.doc.html)
    - [Tutorial](#)
    - [Interactors and Command Objects](#)
  - Novel object, graphics, constraint, input, output, undo, command, and animation models
  - Were widely used for a while





# Garnet, Amulet Design Overview

- Invented our own object system
  - Prototype-instance instead of class-instance
  - Syntax: `prototype.Create("name")`
- **Uses** `obj.set ( instance-variable, value )`
- **Uses** what is now called **method cascading** or **fluent interface**
  - `.set` and other methods return the original object, so can be chained together
  - `Obj.set(Am_X, 4).set(Am_Y, 6).add_part...`
- C++ didn't have name spaces, so started all Amulet words with `Am_ ...`

# Types of Interactors

- Am\_Choice\_Interactor : select one or more of a set of objects
- Am\_Move\_Grow\_Interactor : move or grow objects with the mouse
- Am\_New\_Points\_Interactor: to create new objects by entering points while getting feedback "rubber band" objects
- Am\_Text\_Edit\_Interactor : mouse and keyboard edit of text
- Am\_Gesture\_Interactor: interpret freehand gestures





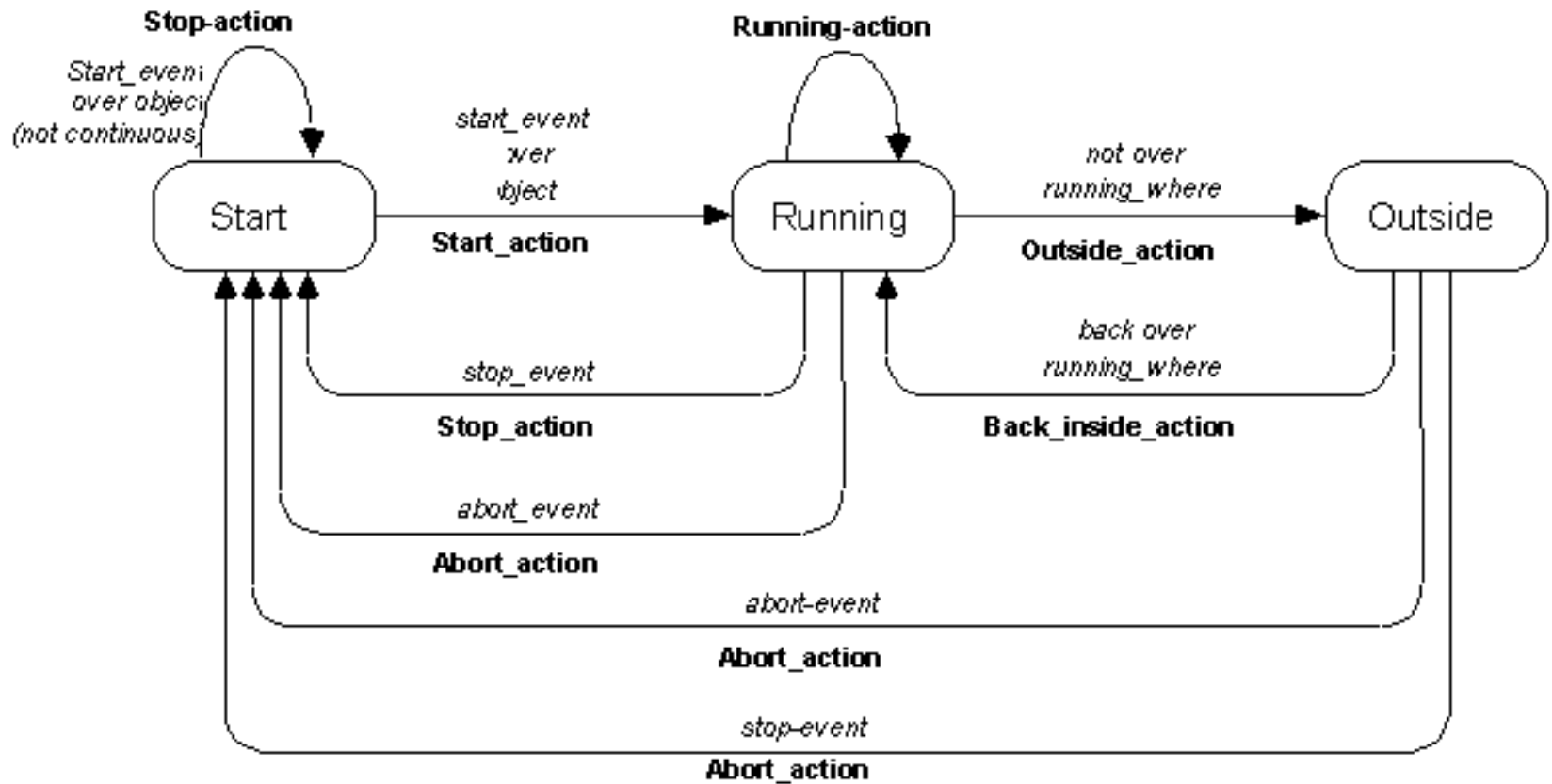
# Affected Graphical Objects

- Set of objects to operate on:
  - To be active, Interactor must be attached to an object which is (recursively) attached to the screen
  - Equivalent to visibility of graphical objects
  - Unlike graphical objects which can only be added as parts of windows or groups, interactors can be added as parts of any object:

```
rect.Add_Part(my_inter);
```
  - Default: operates on the object attached to
  - But also common to operate on any member of a group.
  - Controlled by the `Am_Start_Where_Test` slot, which should contain a method

# Standard Behavior

- (“state diagrams” covered in lecture 9)





# Other standard parameters

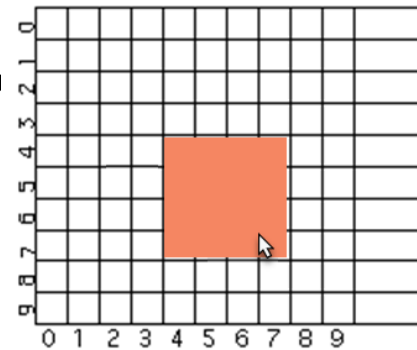
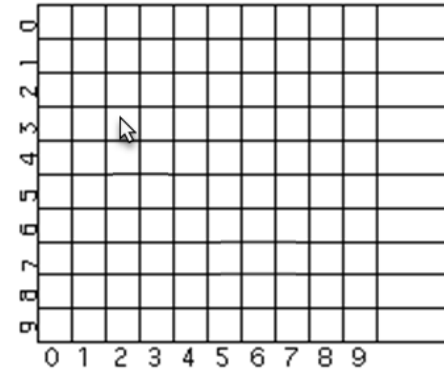
- Multiple groups
  - interactor can span multiple windows
- start, stop and abort events
  - single key, mousebutton, "any" mousebutton, modifiers, (shift, meta...), double click, click vs. drag, etc.
- active?
- priority levels

# Parameters for specific types of Interactors

- For buttons (Choice Interactors)
  - how many objects to select: set, toggle, list-toggle
- For move-grow:
  - interim feedback object (while the mouse moves)
    - if missing then object itself is modified
  - gridding
  - move or grow
- flip if change sides
- minimum size

# Gridding

- Surprisingly complicated
  - E.g., grid of 4 – where is **first point**?
    - $(X+2 \bmod 4)*4 = 4$
  - Should **width** be a multiple of 4 or right side?
    - Width = right side at 7
    - Right side = 8
- Origin of grid – with window or container?



# Flip if change sides

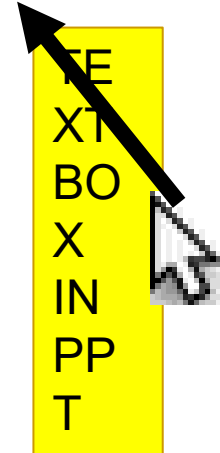
- What happens when move towards upper left?
  - Pegs at minimum size?
    - Most window managers do this
    - Might be zero – shape disappears?
  - Object flips over?
    - PowerPoint does this
    - Text becomes upside down and/or backwards



Test  
Me



TEXT  
BOX  
IN  
ppt



TEXT  
BOX  
IN  
PPT



# Parameters for New\_Point

- interim feedback object (while the mouse moves)
- gridding
- minimum size
- **abort if too small**
  - Avoid creating tiny or invisible objects

# Simple Example

- To make an object movable with the mouse:

```
Am_Object rect = Am_Rectangle.Create() .Set(Am_LEFT, 40)
               .Set(Am_TOP, 50) .Set(Am_FILL_STYLE, Am_Red)
               .Add_Part(Am_Move_Grow_Interactor.Create());
```





# Advanced Feature: Priorities

- If two interactors want to run, priorities used to determine which
- Am\_PRIORITY slot contains a number. Default = 1
- When running, 100 added to it
- Inspector interactors use 300.0
- If multiple with same priority, runs the one attached closer to the leaf

**Lecture 8b:**  
**Output 2:**  
**Basic 2D Computer Graphics**



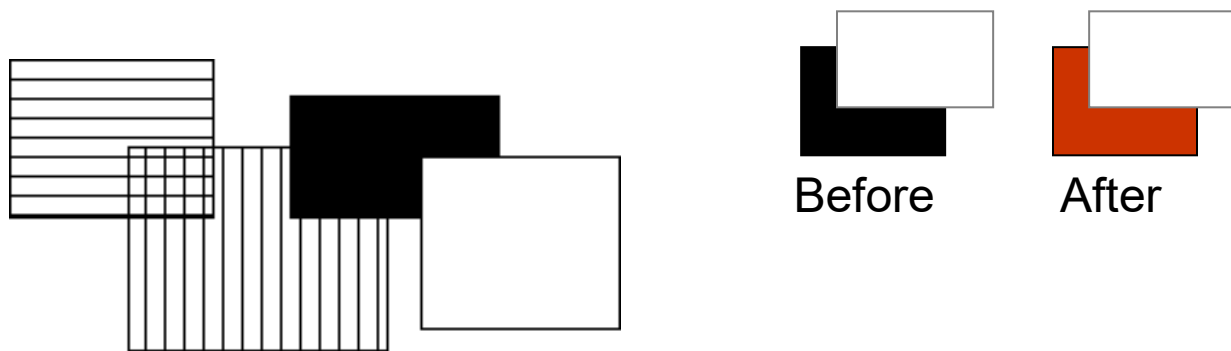
05-431/631 Software Structures for User Interfaces (SSUI)  
Fall, 2022

# DOM is an Example of: Structured Graphics

- Saves a list of all the graphical objects
  - Edit the screen by editing the saved list
- Also called "display list" or "retained object model"
- Provided by many toolkits and graphics packages early vector displays
  - CORE (~1977), GKS (1985), PHIGS (1988)
  - Optional in InterViews, CLIM, etc.
  - Required in Amulet, Garnet, Rendezvous, etc.
- Also SVG that is part of DOM

# Structured Graphics, cont.

- Advantages:
  - Simpler to program with: don't call "draw" and "erase"
    - Just add and remove objects
  - Automatic refresh of windows when uncovered, etc.
  - Automatic redisplay of objects when change and also of other overlapping objects



# Structured Graphics Can Support

- Ability to support:
  - high-level behaviors like move, grow, cut/copy/paste, etc.
  - high-level widgets like selection handles
  - constraints among objects
  - automatic layout
  - grouping: "Groups" in Garnet
  - automatic printing
  - external scripting, ...
  - accessibility

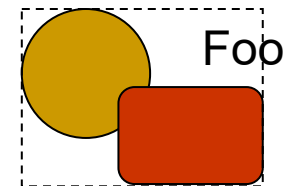


# Structured Graphics Disadvantages

- Disadvantages:
  - Significant space penalties
    - objects take up to 1000 bytes each
    - imagine a scene with 40,000 dots (200x200 fat bits)
  - Time penalties
    - Redisplay doesn't take advantage of special properties of data:
      - regularity
      - non-overlapping
    - Reason for special parameters in lists in React (see future lecture!)

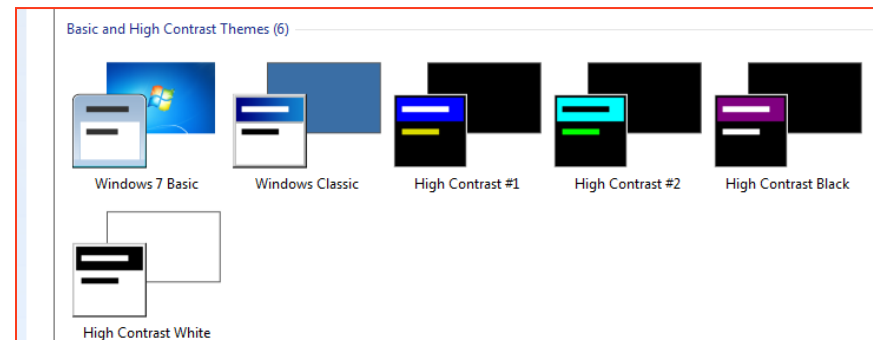
# Basic Idea: Graphical objects retained in a hierarchy

- Primitives: text, rectangles, circles, ...
- **Groups**
  - Also called “**aggregates**”, “**collections**”, ...
  - In HTML: `<div>`, `<ux>`, ...
  - SVG: “**Group**”
  - The size of a group includes all of its “**children**” objects.
    - Also called “**components**”
    - Bounding box of group
  - Group is “**parent**”, elements are “**children**”



# Design Issues: Hierarchies & Inheritance

- How many hierarchies for OO graphics systems?
  - **Inheritance** (class-instance or prototype-instance)
  - **Components / Groups**
  - **Style hierarchies**, like from CSS classes or Windows themes
- Where do properties come from?
  - Color, size, shape
    - From aggregate or inheritance hierarchy?
  - Issue: changing type of object – rectangle → polygon
  - Windows widget properties
    - Size, color scheme, transparency, ...





# Redisplay Algorithms

- Redisplay everything each time
  - Most appropriate for small numbers of objects, and if drawing is really quick compared to computation
  - Used on the Macintosh and many others
  - Used by Amulet
  - I don't know what browsers do

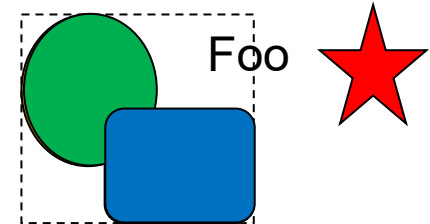
# Redisplay only the affected areas of the screen

- Requires computing what areas are affected
- Garnet:
  - keep track of objects that change any "interesting" slot
  - compute the bounding box of all these changed objects in their old and new locations
  - assert this as the clipping region (must not self-intersect; Garnet uses 2 regions)
  - erase the area
  - go through objects from top-to-bottom, back to front draw those which overlap the bounding box
  - goes through all top level aggregates, and any children of the aggregates that intersect (recursively)
- Other techniques: quad trees



# Overview of Redisplay Algorithm

- Simplest algorithm: draw all objects from back to front
- More sophisticated: Can **clip** to boundary of changed objects
  - 1st pass – collect all the objects which have changed
    - Combine into one or more **clipping** rectangles
  - 2nd pass – go through all objects from back to front and redraw them
    - Will be clipped to affected regions
    - Optimization – only do components if group intersects changed area
  - Issue: complexities determining bounding boxes due to anti-aliasing, miter for polygons, etc.





# Object-Oriented Techniques

## ● Motivation

- Became popular along with GUIs, Direct Manipulation
- Icons, graphics seem like objects:
  - have internal state, persistence
- OO was originally developed (SmallTalk) and became popular (C++) mostly due to GUIs.
- C++ became popular with Windows programming



# Object Oriented

- As a UI technique:
  - Same as GUI, Direct Manipulation = icons, graphical objects, widgets
- Here, as a programming paradigm (often in a language)
- A form of "data abstraction"
- "Classes" describe the basic structure of the data
- Also, the methods that can be called
- Usually no direct access to the data, only the methods

# OO

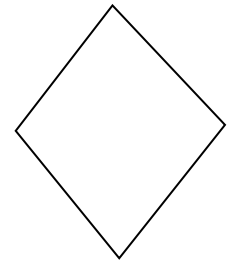
- Create "instances" of the classes
  - local copy of data
  - may also be class data -- all instances share the same value
  - shares all methods
- "Inheritance": create a new class "like" the superclass
  - by default has all the same methods and data
  - can add new data and methods and re-program inherited methods
- Example: `graphical_object.draw ... circle.draw`

# OO

- New style of programming; thinking about the problem
  - Many books about how to do it right.
  - OO design; getting the classes and protocols right
    - So subclasses don't have extra, wasted data space
    - Methods make sense to all sub-classes
    - So external classes don't need to know inside description.
  - Also OO databases, etc.
- Implementation:
  - object in memory, starts with pointer to table of methods, etc.
  - lots of tricks and extra declarations in C++, Java etc. to avoid overhead of lookups at run-time ("virtual", "pure virtual")

# Multiple inheritance

- Class has multiple parent classes
- Combine all the methods and data of all
- Special rules for when conflict (same method, same name of data with different types, etc.)
- Example: circle inherits from graphical-object and moveable-object
- Complex so often not used even when available
  - “Diamond problem”
- Amulet uses constraints to provide flexible copying of values instead
- Java, etc. use “interfaces”
  - No inheritance of implementations, but ability to have arbitrary “mix-ins”
  - No confusion about which superclass to inherit from





# Prototype-Instance model

- Instead of the class-instance model
- All objects are instances
- Can use any object as a prototype for other objects
  - Inherits all slots it doesn't override (= instance variables, member variables, fields, attributes).
  - Methods are just a value in a slot
  - Dynamic changing of methods
- Easy to implement using structures.
- Usually, changing prototype data also changes all instances that do not override it.
- Now used by JavaScript
  - Older uses: ActionScript (Flash), SELF, NewtonScript,

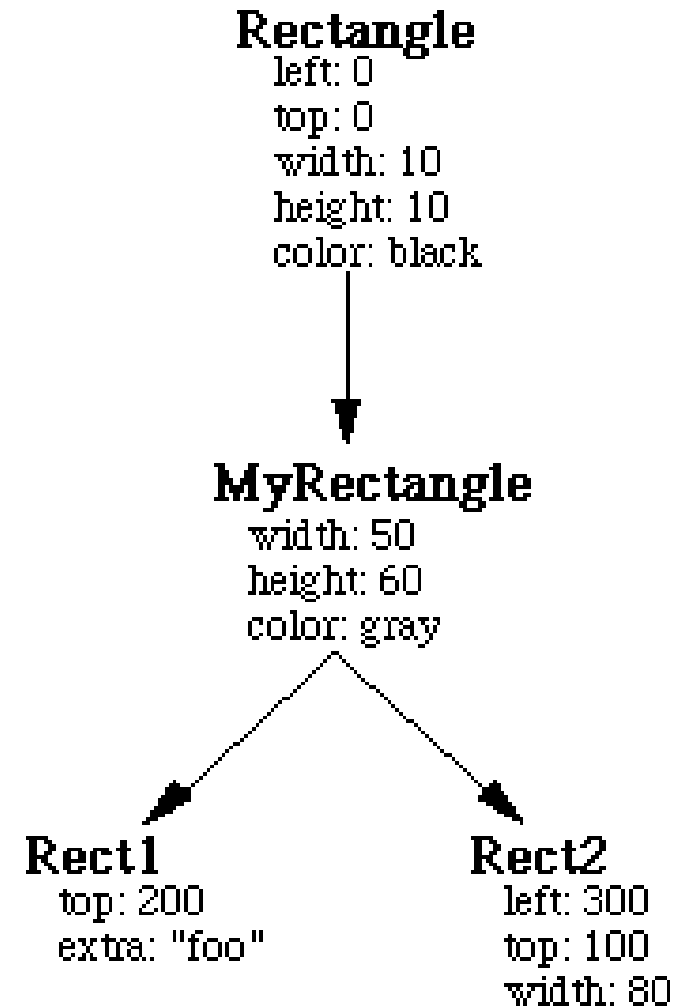


# Prototype-Instance model

- Adding and removing of slots **dynamically** to any instance
- Simpler model, easy to implement
- More dynamic
- But much less efficient
  - Can't usually compile slot accesses into structure access; may need a search
  - No type checking on slots
  - Methods looked up at run-time
  - Space for names of slots, extra pointers, etc.

# Prototype-Instance model

- Available, but not frequently used in JavaScript



# JavaScript class and superclass, and dynamic setting

- ```
class x extends y {  
  constructor (a,b) { //class constructor  
    super(a);      // call constructor of y (required)  
    this.b = b;  
    //other set up stuff  
  }  
  doSomething(a) { //overrides this method  
    super(a); // call y's version of doSomething  
    // local stuff  
  }  
}
```
- **Dynamic creating of new field:**  

```
let myx = new x(4, 5);  
myx.newthing = 6; //creates and sets a new field in myx  
...  
let p = myx.newthing + 12; //somewhere else can use it
```